

Broadcast Federation: An Application-layer Broadcast Internetwork

Yatin Chawathe
AT&T Labs—Research
Menlo Park, CA
yatin@research.att.com

Mukund Seshadri
University of California, Berkeley
mukunds@cs.berkeley.edu

ABSTRACT

Researchers and commercial developers have proposed several protocols to enable efficient multi-point communication both at the IP layer and at the application or overlay layer. However, no single protocol has made enough headway in terms of deployment for it to span the entire Internet. In fact, we believe that none of the existing multicast or broadcast protocols will become the sole dominant Internet broadcasting technology any time in the near future. Instead, we expect islands of non-interoperable broadcast connectivity. To address this, we propose an architecture that enables the composition of different broadcast-capable networks, each with its own broadcast protocol, to provide an end-to-end broadcast service. We call this architecture the *Broadcast Federation*. In this paper, we describe the architecture along with a prototype implementation and preliminary results from the prototype.

Categories and Subject Descriptors

C.2.5 [Local and Wide-area Networks]: Internet; C.2.4 [Distributed Systems]: Distributed applications

General Terms

Design

Keywords

Broadcast, multicast, internetworking, inter-domain, overlay.

1. INTRODUCTION

One-to-many and many-to-many broadcasting¹ applications such as live Internet streaming, multimedia conferenc-

¹In this paper, we use the term “broadcast” to mean one-to-many or many-to-many transmission, instead of the traditional term “multicast”, which we use to refer specifically to the network-layer IP multicast protocol [6].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NOSSDAV’02, May 12-14, 2002, Miami, Florida, USA.
Copyright 2002 ACM 1-58113-512-2/02/0005 ...\$5.00.

ing, and online software distribution are gaining in popularity. These applications are experiencing increasingly large demand driven in part by a growing user population with broadband Internet connections. As an example, a recent online Madonna concert recorded nine million clients [5]. Applications that have such a large client population need efficient one-to-many (or many-to-many) communication primitives within the network.

For over a decade, researchers have spent considerable effort on the design of protocols to support such efficient broadcasting. The IP Multicast service [6] was proposed as an extension to the Internet architecture for efficient multi-point packet delivery at the network level. More recently, in part to address the shortcomings of IP Multicast, researchers and commercial vendors have developed other network layer protocols such as Source-specific Multicast (SSM) [1] as well as a variety of application-layer overlay protocols such as End-system Multicast [12], Scattercast [3], and Overcast [14]. SSM restricts the multicast service model to a single source per session thereby eliminating or simplifying many of the hard multicast problems such as address allocation, access control for sources, and inter-domain routing. Overlay multicast solutions make deployment of broadcast functionality easier since they implement their functionality entirely at IP hosts and require no modifications to the core routing technology.

With this range of broadcast-capable technologies, we have ended up with a scenario where a number of diverse and non-interoperable protocols co-exist in the Internet. No single protocol has been deployed globally. In fact, we do not expect any single broadcast protocol to take over as the dominant Internet broadcasting technology any time in the near future. This is due to a variety of reasons including technical shortcomings in the protocols and their implementations, the fact that some of these protocols are geared toward specific applications, and a range of business model concerns. Instead, the Internet landscape is likely to be fragmented into potentially overlapping clouds of broadcast connectivity with no interoperation across these clouds.

To address this situation, we propose an architecture called *Broadcast Federation* for composing the broadcast connectivity offered by different broadcast networks, each of which implements its own independent broadcast protocol. Our architecture interconnects individual broadcast-capable networks, such as a single ISP’s IP multicast network or a Content Distribution Network spanning multiple ISPs, into a federation of loosely-coupled broadcast systems. The architecture is depicted in Figure 1. At its core is the *Broadcast*

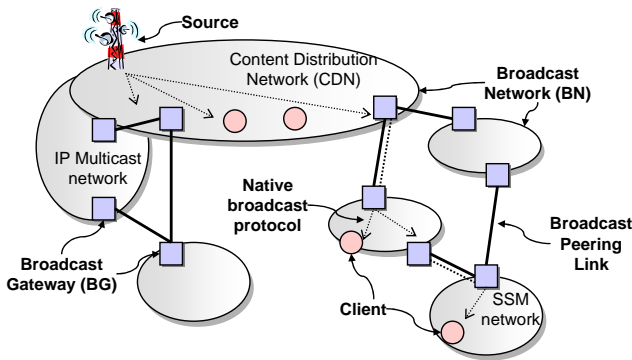


Figure 1: The Broadcast Federation Architecture. *Broadcast Gateways* are explicit application-layer peering points that interconnect a diverse collection of *Broadcast Networks* and translate between the inter-network protocols, and the individual native intra-network protocols.

Gateway (BG), which is the point of peering between different *Broadcast Networks (BNs)*. Within a single BN, the architecture uses the native broadcast protocols of that network, while BGs act as translation and forwarding points across BNs. Peering relationships between broadcast gateways are administratively configured.

BGs construct an overlay internetwork across the federation using a suite of protocols: (1) a routing protocol which propagates reachability information across BNs, (2) a tree-building protocol which constructs overlay distribution trees across the federation, and (3) a data forwarding protocol which sets up transport channels for transmitting data packets along the distribution trees. These protocols apply the principles of network-layer routing protocols such as BGP [17] and PIM [7] across the entire federation.

To interoperate across a range of native broadcast protocols, each broadcast gateway has a customizable component called the *NativeNet (Native Network interface)* which isolates the implementation of the native broadcast capability for that BN. Each broadcast gateway implements a custom version of this component to establish a mapping between the federation protocols and the specific protocols used within its own native broadcast network.

The rest of the paper is organized as follows. In Section 2, we present relevant background and related work. Section 3 describes the components and service model of our architecture. In section 4, we present the details of the federation protocol suite. Section 5 describes our prototype implementation and demonstrates how the federation components are customized for specific broadcast networks. Finally, we present on-going research in Section 6 and conclude in Section 7.

2. BACKGROUND AND RELATED WORK

The Internet community has tried to solve the problem of providing an efficient multi-point data delivery framework for a long time. Deering et al proposed IP multicast [6] as an extension to IP that provided routing-layer support for many-to-many packet delivery. In this approach, a source sends packets to a multicast *group* address, and the routing infrastructure deals with the details of forwarding those

packets to all receivers in that multicast group. Although in theory an IP-multicast-enabled Internet can provide optimal delivery for broadcast packets, IP multicast has failed to take off in the deployed Internet. In [8], the authors describe many of the concerns such as scalable inter-domain multicast, distributed address allocation and access control, that have confounded ISPs while deploying IP multicast technology. Partly in response to these concerns, researchers and commercial developers have introduced new multi-point communication technologies that address some of the problems associated with IP multicast.

SSM (Source-specific Multicast) [1] changes the IP multicast service model from a many-to-many communication primitive to a one-to-many primitive. SSM permits a single explicit source within each SSM group. In doing so, it eliminates the problems of address allocation, access control and scalable inter-domain rendezvous and routing that plague IP multicast. However, like IP multicast, SSM is network-layer technology, and although router support for SSM is available, few ISPs offer it as a widely-available service.

Recently, there have been proposals for moving the multi-point communication functionality away from network routers into application-layer overlays. The End-system Multicast project [12] builds broadcast functionality entirely out of the end-hosts that participate in a broadcast session. The Scattercast [3], Overcast [14], and Inktomi Broadcast Overlay [9] architectures build a broadcasting service via specialized broadcast servers that form overlay networks of unicast connections on top of the existing IP networks. By relying entirely on commodity hardware and application-layer software, these systems make deployment substantially easier, but at the price of less-than-optimal distribution trees.

To our knowledge, there has been no significant effort at bridging different multicast and broadcast networks into a single cohesive internetwork. Some content-distribution architectures such as Scattercast and the Inktomi Broadcast Overlay use a combination of unicast and IP multicast within their overlays, but they cannot interact with each other or with other broadcasting technologies.

In the context of the web, we have seen proposals [13, 2] for peering between independent Content Distribution Networks (CDNs). CDN peering provides CDNs a way to interoperate by allowing them to intelligently redirect clients to one another. Like CDN peering, the Broadcast Federation allows independent broadcast providers to interoperate. However, unlike CDN peering, the federation architecture needs to deal with live distribution of broadcast content across providers rather than simply redirecting clients from one CDN provider to another.

The Broadcast Federation is essentially an internetwork architecture between independent broadcast service providers. This is akin to the inter-domain IP multicast problem, which seeks to interconnect independent multicast-capable ISPs across the Internet. For this purpose, multicast providers use PIM [10] as the multicast protocol within their networks and MSDP [16] to interconnect these individual networks. However, MSDP cannot scale to an Internet-wide deployed solution since it floods information about all active sources across the entire Internet. The Border Gateway Multicast Protocol [15] improves upon MSDP by proposing a solution which is independent of the intra-domain multicast protocols. This solution is based on (a) defining the concept of a root domain to be associated with each multicast group,

(b) using multi-protocol extensions to BGP [18] to distribute “group routes”, the shortest paths toward the root domain for each multicast group, and (c) building an inter-domain reverse-shortest-path tree based on explicit join messages toward the root domain for each multicast group. The Broadcast Federation architecture leverages these design ideas, but extends them beyond the IP multicast framework to apply to a heterogeneous collection of network- and application-layer broadcast technologies.

3. THE FEDERATION ARCHITECTURE

Figure 1 shows the essential components of the Broadcast Federation architecture. A federation is composed of a collection of *Broadcast Networks (BNs)* that are interconnected via an overlay of *Broadcast Peering Links* across application-layer *Broadcast Gateways (BGs)*. We now define each of these elements.

3.1 Architecture Components

Broadcast Network (BN): A BN is a network which supports a single broadcast or multicast protocol natively and is bounded administratively and in scope. All nodes within a BN use the native broadcast medium to transmit or receive content local to their BN. The native broadcast medium may be based on either a network-layer protocol such as IP multicast [6] or Source-specific Multicast (SSM) [1], or an application-layer broadcast Content Distribution Network (CDN) [3, 9, 14].

A BN may be restricted to a single ISP’s domain, or may span multiple ISPs as in the case of a CDN such as the Inktomi Broadcast Overlay [9]. Each BN is identified by a globally unique 32-bit BN number (*BNNum*). We rely on DNS to convert descriptive BN names into BN numbers. For example, a well-known IP address within the BN and the DNS name associated with that IP address can be used to represent the BN’s *BNNum*.

Broadcast Gateway (BG): The broadcast gateway is the point of peering between BNs, and implements the bulk of the federation protocol suite. Each Broadcast Gateway has a module called *NativeNet*, which implements the customization components that are required to allow the BG to communicate with its native broadcast medium. It translates incoming content and control messages into appropriate messages for the native medium.

Conceptually, the BG is a single node in the architecture. In practice, for scalability and availability, it may be implemented as a cluster of cooperating machines [4, 11].

Broadcast Peering Link: A peering relationship between two BGs indicates a willingness to carry broadcast traffic to and from each other’s networks. BGs across individual broadcast networks establish administratively configured peering links to each other (in contrast to self-organizing peers). This allows BN administrators to establish explicit policies and commercial agreements for the use of those links. For example, BN administrators can configure peering links to allow or disallow the use of their network as a transit for traffic to and from other reachable networks. The peering connections that are set up across BNs are known as “external” peering links. In addition, all BGs within a single BN set up “internal” peering links with one another.

3.2 Service Model

The federation service model defines a broadcast *session*

as a federation-wide group-communication abstraction. Clients across the entire federation can subscribe to sessions published anywhere within the federation. Thus, a federation session extends an IP multicast network’s group abstraction or a single CDN’s stream beyond that network’s boundaries to reach clients in other BNs.

The network in which a broadcast session is originally published is designated the *owner BN* for that session. Typically, the owner BN is also the network in which the primary source of content for the session is located. For example, an online music concert that is broadcast via an ISP’s local IP multicast medium will choose that ISP’s network as the owner BN for the concert. Clients within the owner BN can directly subscribe to the session without interacting with any of the federation components. This ensures that the native service model for a BN remains unaffected by the fact the the BN is part of a wider federation. Thus, in our online concert example, clients within the owner ISP’s domain can tune into the music concert merely by subscribing to the IP multicast group for the concert. Beyond the owner BN, clients will rely on federation protocols to access the session via their broadcast gateways.

The owner BN provides a convenient rendezvous point for the session. Other broadcast networks attach to the session via the owner BN’s broadcast gateways. Moreover, the concept of the owner BN lends itself to an intuitive naming scheme for broadcast sessions that avoids the address allocation problems associated with traditional multicast protocols. Across the federation, a session is identified by a combination of the owner BN’s *BNNum* or its corresponding DNS name, and the native name of the session within the owner BN. We use a URL-like naming scheme for these federation-wide session names:

```
bfed://<owner-BNNum-or-name>/<native-name>
?<pmtr1>=<val1>&<pmtr2>=<val2>...
```

The *<native-name>* component of the session name represents the address or unique name associated with the session within the owner BN, and can be parsed only by gateways in the owner BN. Clients and gateways in other BNs regard it as an opaque string of characters. Thus, the music concert originating within an ISP’s IP multicast network will use the *BNNum* or DNS name associated with the ISP’s network, and the IP multicast group address to form its federation-wide session name, for example:

```
bfed://multicast.isp.net/224.5.6.7:8888
```

In addition to the owner *BNNum* and the native name, a session URL may contain additional parameters. These parameters, which follow the “?” symbol in the URL, specify options for the session that allow the federation to support a wide range of session types and broadcast network capabilities without imposing a lowest-common-denominator service model across all networks. Thus, the federation can identify sessions as single-source or multi-source, real-time or bulk-data, etc. by specifying appropriate options within the session URL. The entire session URL including the owner *BNNum* or name, the native session name, and the parameter list together form a unique session name across the federation. For example:

```
bfed://multicast.isp.net/224.5.6.7:8888?content=
real-time&single-source=true
```

4. THE FEDERATION PROTOCOLS

We now describe the protocol suite that forms the basis for unified communication across the broadcast federation. The design of our protocols is guided by two principles:

- Loosely-coupled interaction across BNs, rather than tight integration of each BN's native broadcast protocols into the federation.
- The ability of each BN to publish sessions for its own clients without requiring any intervention from the federation. The federation protocols should be invoked only when clients need to access sessions across BNs.

The federation protocol suite is composed of three tiers: a routing layer that propagates reachability information across BNs, a tree-building layer that constructs distribution trees across the federation, and a data forwarding layer that handles the distribution of data packets across the dynamic trees constructed by the tree-building layer. These protocol layers allow a session to be disseminated via distribution trees that are constructed from the reverse shortest routing paths between each destination BN and the owner BN of the session in a manner similar to that used by PIM [7] for transmitting IP multicast data. Each of the protocol layers has policies and mechanisms that need to be customized for each specific BN and its native broadcast medium. These customizations are encapsulated within a layer called NativeNet (*Native Network* interface). We describe the three core protocol layers and the customization layer in the following sections.

4.1 Routing Layer

Akin to BGP [17], the routing layer uses a path-vector protocol to propagate information on how to reach other BNs to all broadcast gateways in the system. Each broadcast gateway maintains a routing table that is used to compute the best route from a client's BN to any other BN in the federation. This simplistic approach would result in aggregating route information on a per-BN basis, that is, all content from a specific source BN to a specific destination BN would always use the same route. This is undesirable when an owner-BN has more than one broadcast gateway. Consider, for example, a large BN, say BN_1 , with two broadcast gateways which are topologically distant from each other. If all routing information is aggregated at the BN level, then all sessions from BN_1 to another network, BN_2 , would be routed through only one of the two broadcast gateways that BN_1 has. In reality, the best route for a specific session originating within BN_1 would depend upon the proximity of the source of that session to each of BN_1 's gateways. Hence, the routing protocol should be able to pick the appropriate route on a per-session basis.

On the other hand, to ensure that per-session information is not sent to all BNs across the entire federation (many of which many not be interested in all of that information), the routing layer itself should be session-agnostic. That is, the routing layer should not distribute any information about individual sessions. This is key to ensuring that the routing infrastructure can scale independent of the total number of sessions. Hence, to allow us to pick appropriate inter-BN routes on a per-session basis without flooding per-session state across the federation, we aggregate routing information at the gateway level. In other words, the routing protocol advertises routes to individual BGs rather than to entire

BNs. This allows us to pick the appropriate exit broadcast gateway from a session's owner BN on a session-by-session basis. We describe how this is achieved later in Section 4.2.2.

Traditional routing protocols use latency or hop count as the metric for evaluating routing costs. However, the routing metric that is most useful for a specific session depends on the characteristics of the session itself. For example, a delay-sensitive application like video conferencing cares most about the latency metric, while a bulk data transfer application needs routes that provide the best bandwidth performance. Hence, instead of fixing one routing metric for the entire federation, the routing layer presumes no single metric. It constructs a collection of routing tables, one for each metric of interest.

Different routing tables can also be used to encode the diverse capabilities of the broadcast networks themselves. For example, BNs that are not capable of or do not want to support multi-source broadcast distribution can opt out of acting as transit for such sessions by advertising independent routing tables for single-source and multi-source sessions. Thus a BN that uses a Source-specific Multicast (SSM) native medium can advertise infinite cost transit paths in its multi-source routing table, but advertise the actual routing costs in the single-source table. Similarly, we allow BNs to differentiate between their abilities to carry different kinds of traffic. For example, an HTTP-based CDN is not amenable to carrying real-time audio/video data over its TCP-based native broadcast medium, and hence can choose to advertise higher cost for such content.

Sessions indicate their routing requirements via parameters in their session URL. For example, `metric=latency`, `metric=bandwidth`, `multi-source=true`, `content=real-time`, `content=bulk-data`, etc. As we shall see in the following section, the tree-building layer picks the appropriate routing table for its queries based upon these parameters that it extracts from the session URL.

The routing layer provides hooks for implementing policy based on peering agreements across BNs. For example, a BN may not wish to act as a transit network for other BNs. It can do this by applying appropriate filters to the routing information before forwarding the information to its neighboring BNs. This is similar to the policy-based filtering that is used in the Internet today by BGP. The filtering hooks provide a basis for BN administrators to establish different kinds of commercial relationships with each other, like customer-provider relationships or peer-peer relationships.

In summary, the federation routing layer performs path-vector routing across BGs. It allows individual broadcast networks to export different routing costs depending upon individual session requirements and the capabilities of the BNs themselves. The routing information that is exported by a BN can be filtered based on administrative policies and peering agreements that the BN has set up.

The actual messaging protocol between two peer BNs relies on pairwise exchange of route-update messages over TCP connections. The first update conveys the entire routing table after being passed through policy-enforcing filters. Subsequent updates are incremental in nature. BGs use periodic keep-alive messages to ensure that their peers are still available. When a BG discovers the loss of a peer due to an absence of keep-alive messages, it sends to all its other peers route withdrawal notifications for all routes that went

through the lost peer.

4.2 Tree-building Layer

The tree-building layer builds shared inter-BN distribution trees. There is one tree for each session and the tree is rooted at the owner BN of the session. Multi-source sessions do not construct per-source trees. All data traffic for the session flows along the shared tree. The tree is constructed using the reverse shortest paths between BGs of subscribing clients and the appropriate BGs within the owner network. This is similar to the rendezvous-point-rooted trees constructed by PIM [10].

To join a broadcast distribution tree, clients must first communicate with a broadcast gateway within their BN. We introduce the concept of a *Mediator* which intercepts JOIN requests from clients and forwards them to an appropriate broadcast gateway. Once such an access gateway is located, the access gateway must initiate the tree-building process. In order to optimize distribution trees on a per-session basis, the tree-building process consists of two steps: a Session-specific Route Discovery (SROUTE) step, followed by a JOIN step. We examine each of these in detail in the following sections.

4.2.1 Mediator

When a client wishes to participate in a federation session, it needs to locate a broadcast gateway within its BN to which it can forward its JOIN request. However, the manner in which this is achieved in a scalable fashion depends upon the characteristics of the specific BN. For example, clients in an IP multicast BN can use the multicast routing infrastructure to request sessions, while clients in a CDN can request sessions via the CDN's edge servers. Hence, we isolate this functionality into an abstraction that we call a *Mediator*. The mediator is responsible for intercepting JOIN requests from clients and forwarding them to one of the broadcast gateways within the client's BN.

The mediator functionality can be implemented either within the BN's broadcast gateway or as part of the BN's native broadcast network. Each BN in the federation implements its own mediator. In an IP multicast network, the mediator can be a well-known IP multicast group to which clients and gateways subscribe. Clients send their federation JOIN requests to this multicast group, the multicast routers forward the requests across the BN, and the broadcast gateways intercept the requests. Alternatively, in a CDN-based BN, the CDN's edge servers can incorporate the functionality of the mediator. By looking at the URL within the client's request, an edge server can forward the request to the local CDN infrastructure if the request is for a local session, or to one of the local BGs if the request is for a federation session. We describe a more detailed example of how mediators are specialized for specific broadcast networks in Section 4.4.

Clients in the owner BN of a session do not require any mediator. They simply use the owner BN's native subscription mechanisms to participate in the session. When clients need to access a session that does not originate in their own BN, they contact their local mediator and send it a JOIN request that includes the full session URL. The mediator in turn contacts an appropriate broadcast gateway to initiate the tree-building process. The broadcast gateway eventually responds with either a TRANSLATION message

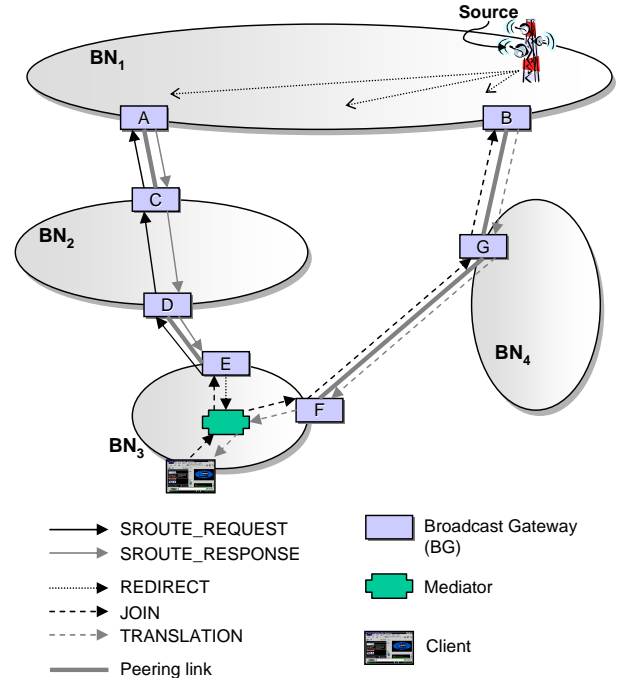


Figure 2: The SROUTE Discovery protocol. A client in BN_3 sends a JOIN request to its BG E via the mediator. E sends an SROUTE_REQUEST upstream toward A . A returns an SROUTE_RESPONSE back to E . E redirects the mediator to the other BG F . The mediator sends a JOIN request to F which forwards the request toward B . Each BG generates TRANSLATION responses for the received JOIN requests.

that contains a locally valid native broadcast address, or a NO_ROUTE message indicating that there is no path from the client's BN to the owner BN for the session. Clients can then directly subscribe to the native address specified in the TRANSLATION response.

In our current architecture, we assume that a client belongs to or has access to a single BN, and is pre-configured with information about the local mediator for that BN. Clients may either be statically configured with this information or may be handed the information via special DHCP options. An alternative approach would be to establish a global mediator service that all clients contact. Such a mediator service would automatically determine the best access BN for each client out of potentially multiple choices, and then redirect the clients to the local mediator for that BN. However, the design of this global mediator service is orthogonal to the rest of the federation architecture, and is the subject of future work.

4.2.2 Session-specific Route (SROUTE) Discovery

When a broadcast gateway receives a JOIN request from a client via the mediator, the BG needs to subscribe to the session via its best path to the owner BN. The JOIN request includes the session's URL name. Encoded within the URL are parameters that determine which routing metric to use for looking up the best path toward the owner BN. However,

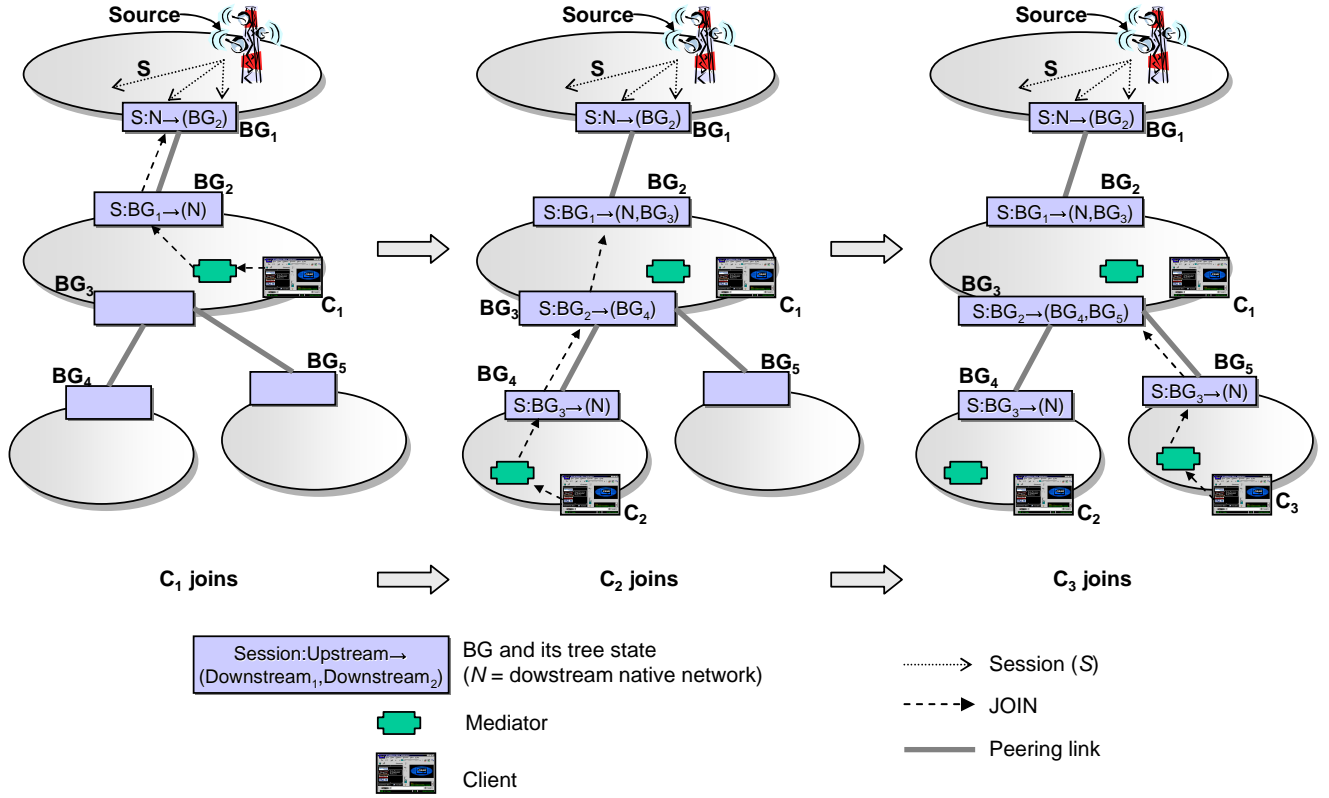


Figure 3: The tree-building protocol and the associated state maintained by BGs participating in the protocol.

the best path may depend upon the location of the session source within the owner BN. As shown in Figure 2, consider an owner network (BN_1) with two broadcast gateways, A and B . Since the routing layer keeps track of BG-BG routes rather than simply inter-BN routes, the broadcast gateway E in BN_3 has two routes $E-D-C-A$ and $E-F-G-B^2$ that lead to BN_1 . Similarly, BN_3 's other gateway, F , has two routes leading to BN_1 : $F-G-B$ and $F-E-D-C-A$. When a client in BN_3 joins a session via its local mediator, the mediator and the local gateways need to determine which of the above routes are best suited for accessing that session. This is achieved through the SROUTE Discovery protocol.

The Discovery protocol works as follows. A mediator intercepts a JOIN request from a client and forwards it to any one of the broadcast gateways within its BN. In Figure 2, the mediator in BN_3 forwards a client's JOIN request for a session originating in BN_1 to its gateway E . E determines that there are multiple alternative paths that can be used to subscribe to this session. So it sends an SROUTE_REQUEST message along the better of its two paths leading to BN_1 . Included in this message is a list of broadcast gateways of BN_1 that E is aware of (i.e., A and B). The message propagates along the path $E-D-C-A$. Once the SROUTE_REQUEST message reaches A , A sends back an SROUTE_RESPONSE. This response contains information about the proximity of the root of the native broadcast tree used within BN_1 to both the gateways A and B . For example, if BN_1 's na-

²This follows from the fact that BGs within a single network all set up internal peering connections to each other.

tive broadcast protocol constructs source-rooted trees, then the SROUTE_RESPONSE would contain the routing cost between the source and each of the gateways A and B . To determine these costs, A may have to internally communicate with the other gateway B to initiate cost measurement processes. The mechanism by which these costs are determined depends on the specifics of each individual BN. As we shall see in Section 4.4, this mechanism is encapsulated within the NativeNet implementation for the BN. If a BN does not wish to expose any of its internals to the rest of the federation, its NativeNet implementation can simply return a null SROUTE_RESPONSE.

Once the SROUTE_RESPONSE returns to BN_3 , E can determine the best route for that specific session. It does so by appending the SROUTE costs to the BG-BG routing costs that it has stored in its routing tables. In our example, E determines that $F-G-B$ is the best session-specific route. So, it sends a REDIRECT response to the mediator indicating that it should subscribe to the session via F . At the same time, it forwards the SROUTE information to F . The mediator in turn sends a JOIN request to F which confirms that it is indeed the best route for this session by using the forwarded SROUTE information. Finally, F sends JOIN messages along $F-G-B$ to subscribe to the session. To avoid extra SROUTE discovery steps by BGs along the JOIN path, the JOIN message also forwards the SROUTE information that F already has.

All of this SROUTE information is “soft” state in that it needs to be periodically refreshed. Every time periodic JOIN messages reach the owner BN, it sends back SROUTE

updates along the join path. Moreover, SROUTE state is maintained only within the set of broadcast gateways that care about it. This is the set of gateways that are part of the distribution tree for the specific session. In our example, the SROUTE state is stored initially along the paths $A - C - D - E$ and $B - G - F$. However, C and D will eventually time out this state since the JOIN messages for this session (which cause the SROUTE state to get updated) flow along the $F - G - B$ path. Other gateways in BNs that do not participate in this session will never receive or store the SROUTE information.

The SROUTE Discovery protocol is performed only when a downstream broadcast gateway has paths to at least two different BGs in the owner BN, and the next-hop peers for those paths are not the same. If there is a single unambiguous next hop for all paths leading from the downstream BG to the owner BN, it can directly send its JOIN message to this next hop peer without initiating any SROUTE discovery.

4.2.3 Tree Setup

Federation distribution trees are per-session shared trees rooted at the owner BN of the session. Downstream nodes send explicit JOIN messages along their best path toward the owner BN as computed by the SROUTE discovery process described above. Once a BG locates the appropriate route, it forwards the JOIN request to the next-hop BG along that route. The JOIN request is forwarded along next-hop BGs toward the owner BN until the request reaches a BG in the owner BN or a BG which is already part of the distribution tree for that session. If this final BG is in the owner BN, it then subscribes to the native session identified within the session URL.

Each BG along the distribution tree maintains tree state that includes the name of the session, the upstream peer in the tree leading toward the owner BN, and a list of downstream peers. Figure 3 shows an example of the tree-construction mechanism and the state that each BG maintains. Like the SROUTE state, this state is soft in nature and the BGs periodically update it by sending refresh JOIN messages upstream. When a BG stops receiving JOIN refreshes from any of its downstream peers, or if it receives an explicit LEAVE message, it removes the tree state associated with that peer. When it has no more downstream peers or clients for a session, it in turn sends a LEAVE message to its upstream peer.

4.3 Data Forwarding Layer

This layer of a BG is responsible for setting up data communication *channels* corresponding to the edges of the distribution tree, and for forwarding packets via those channels. A channel is an explicit communication link that the data forwarding layer within a BG sets up for each of the sessions flowing through that BG. Across external peering links, data forwarding occurs via unicast channels, while within a BN or between internal peers, the forwarding channels use the native broadcast medium. For unicast communication, the BGs use UDP for sessions with real-time or best-effort requirements, or TCP for bulk data transfer sessions.³

³TCP provides hop-by-hop reliability and congestion control between BGs for bulk-data applications. However, to achieve end-to-end reliability or congestion control, applications would have to build additional support on top of this framework.

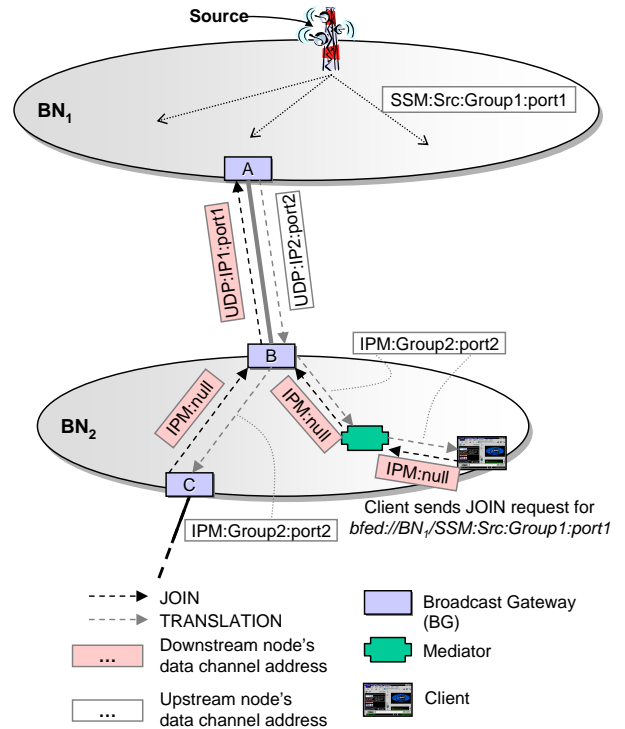


Figure 4: TRANSLATION messages exchanged to set up the data forwarding channels.

To set up these communication channels, the members of the broadcast distribution tree exchange TRANSLATION information during the JOIN phase to convert the federation-wide session name into locally valid names or addresses for each link along the tree. As the JOIN request propagates upstream, each downstream node allocates a communication socket for receiving data from its upstream peer. If the upstream peer needs to know the address of this communication socket before it can send data to it, as in the case of unicast UDP channels, then the downstream node includes the address as a TRANSLATION field in its JOIN request. The upstream node in turn allocates its own communication socket and returns an explicit TRANSLATION message that contains the address of the upstream socket. This is depicted in Figure 4 where a client in an IP multicast network BN_2 requests a Source-specific Multicast (SSM) session originating in BN_1 . When the client issues a federation JOIN request via its local mediator to a BG B in its network, B allocates a unique IP multicast group address from a pool of reserved addresses and returns it to the client via the mediator in a TRANSLATION response. B in turn forwards the JOIN request to its next hop gateway A toward the owner BN. Along with the request, it includes in a TRANSLATION field the IP address and port number of a unicast UDP socket that it allocates for receiving data for this session from A . When A receives the JOIN request, it allocates its own unicast UDP socket for data communication with the downstream peer B and returns a TRANSLATION response that includes the IP address and port number of the socket. Once the JOIN request reaches the owner BN (through gateway A in our example), the gateway subscribes to the native SSM session that is specified as part of the session URL. Later, if

```

class NativeNet {
public:
  /* upcalls */
  void *allocate_channel(SessionDescription *session,
    Boolean up_or_down, char *&ret_addr, int &ret_len);
  void subscribe(SessionDescription *session,
    Boolean up_or_down,
    void *channel, char *addr, int len);
  void refresh(SessionDescription *session,
    Boolean up_or_down, void *channel);
  void unsubscribe(SessionDescription *session,
    Boolean up_or_down, void *channel);
  void reclaim_channel(SessionDescription *session,
    Boolean up_or_down,
    void *channel, char *addr, int len);
  int send_data(void *channel, char *data, int length);
  List *get_sroutes(SessionDescription *session,
    List *sroute_requests, Boolean *got_all_ptr);

  /* downcalls */
  void recv_join_request(SessionDescription *session);
  void recv_leave_request(SessionDescription *session);
  void recv_data(void *channel, char *data, int length);
}

```

Figure 5: NativeNet Abstract API

the gateway C in BN_2 receives a JOIN request for the same session from an external downstream BN, it forwards the request to B , which responds with a TRANSLATION message. This message includes the same IP multicast group address that B had previously allocated for this session.

Forwarding of data packets takes place along these channels as they get set up along with the JOIN messages that propagate up to the owner BN. Distribution trees are assumed to be uni-directional unless a session is specifically flagged as multi-source (via a parameter in the session URL). For uni-directional trees, a BG forwards a packet to all of its downstream tree neighbors only if it was received from the BG's upstream neighbor in the tree. For multi-source sessions, the trees are bi-directional and a BG forwards packets received from any of its tree neighbors to all of its other tree neighbors.

All of the data channels rely on soft state. The periodic JOIN messages generated by the tree-building layer refresh the data channels. If the JOIN messages stop arriving, or if an explicit LEAVE message is received, the channel state is torn down. Moreover, if the routing state changes, resulting in a corresponding change in the distribution tree, new JOIN messages received along the new distribution paths will automatically trigger the set up of new data channels for the updated tree.

4.4 The NativeNet Layer

In the previous sections, we have assumed that the various layers of the broadcast gateway know how to communicate with the gateway's native network, for example, allocating addresses in the native broadcast medium, sending and receiving packets on the native medium, etc. Instead of exposing each broadcast network's functionality to all of the other layers of the BG, we encapsulate this functionality within a simple and explicit customization interface that we call the Native Network Interface or *NativeNet*. Figure 5 shows the

abstract API for this customization interface. Each broadcast network must provide its own implementation of this API. The NativeNet API consists of seven major upcalls that various layers of the BG make to the NativeNet implementation, and three downcalls from the NativeNet to the other BG layers. We describe these briefly below:

allocate_channel: Invoked when the BG needs to allocate a locally-scoped native broadcast address for a new federation session.

subscribe: Used to subscribe to the native broadcast channel that was allocated for a specific federation session. The implementation of this upcall should set up state to allow the BG to listen for arriving data packets on the subscribed channel and hand those data packets back to the data forwarding layer.

refresh: Invoked periodically to allow the NativeNet to refresh its subscription to the native broadcast channel if necessary.

unsubscribe: Invoked when the channel is no longer needed.

reclaim_channel: Invoked to allow the NativeNet implementation to reclaim previously allocated channel addresses that are no longer needed.

send_data: Used to send data packets to the native broadcast medium.

get_sroutes: Invoked in the BG of an owner BN to retrieve SROUTE information from the BN.

recv_join: Invoked by the NativeNet whenever it receives a JOIN request from a client via the local mediator. Depending upon the NativeNet implementation, the mediator functionality may be part of the NativeNet itself or may be implemented in some other component of the native broadcast network. The `recv_join()` downcall in turn notifies the tree-building layer that a JOIN request was received from a client.

recv_leave: Invoked by the NativeNet when it no longer has clients within the broadcast network that are interested in a previously subscribed session. This in turn notifies the tree-building layer to initiate upstream LEAVE requests if necessary.

recv_data: Invoked by the NativeNet whenever it receives data from a native channel that it had previously allocated. This method hands the received data to the data-forwarding layer which in turn sends it on to the rest of the federation.

To understand how this NativeNet API is customized for specific BNs, let us consider an IP multicast BN. The IP multicast NativeNet uses a pre-assigned pool of IP multicast group addresses to allocate to data channels for requested federation sessions. The `allocate_channel()` call assigns an unused address from this pool. The `subscribe()` call opens a datagram socket and connects the socket to the allocated IP multicast group. Since the IP multicast socket implementation automatically deals with refreshing the multicast join state, the `refresh()` call has an empty implementation. `unsubscribe()` closes the multicast socket, and `reclaim_channel()` returns the assigned multicast group address back to the unused pool of addresses. The `send_data()` call is used to send data to the native IP multicast group. If a data packet is received from a previously subscribed IP multicast group,

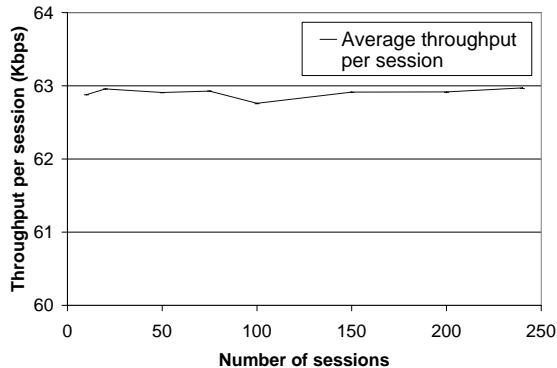


Figure 6: Throughput through a pair of BGs when source is rate-limited to 64Kbps.

the packet is handed down to the data-forwarding layer via the `recv_data()` downcall.

The multicast NativeNet implementation uses a well-known IP multicast group to support the mediator functionality. Clients send JOIN requests for federation sessions to the mediator group. To prevent an implosion of JOIN requests, clients damp their requests by listening for requests from other clients and suppressing their request if they hear another request for the same session. BGs respond to JOIN requests with TRANSLATION messages that contain an IP multicast group address allocated for the session by the BG. If multiple BGs within the BN can respond to a JOIN request, the BG with the shortest (SROUTE-augmented) path to the owner network wins. The mediator implementation interacts with the rest of the BG via the `recv_join()` or `recv_leave()` downcalls. When the mediator receives JOIN requests from the mediator multicast group, it invokes the `recv_join()` call. Once JOIN requests for a session stop arriving, the mediator invokes the `recv_leave()` call to notify the tree-building layer to tear down the state for that session.

Finally, for sessions that are owned by the IP multicast network, the BGs may invoke `get_sroutes()` to determine session-specific routes. Assuming a PIM-based IP multicast network, the SROUTE response will have to contain distance information between the BGs and the PIM rendezvous point for the session in question. Depending on the configuration of the BN, this information may or may not be easily accessible. For example, if rendezvous points are statically configured, the BGs can use this knowledge to perform “ping” experiments to the appropriate rendezvous point for the session. Alternatively, the NativeNet implementation can include a PIM speaker [10] that talks the PIM protocol with neighboring PIM routers and figures out the location of the PIM rendezvous points in that manner. However, a simple BN implementation, or a BN that does not wish to expose its internal routing information to the rest of the federation, can simply choose to return null SROUTE responses. In that case, the `get_sroutes()` implementation will be empty.

5. IMPLEMENTATION

We have built a prototype implementation of the broadcast gateway as a single-machine C++ application. To demonstrate the flexibility of the customization interface, we have implemented NativeNets for three different networks: IP

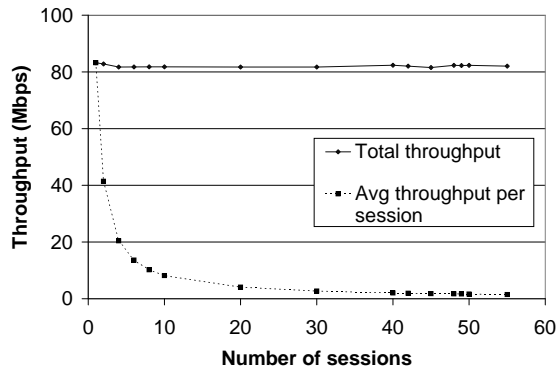


Figure 7: Throughput through a pair of BGs when source transmits at maximum possible rate.

multicast, a single-source SSM network, and an application-layer HTTP-based Content Distribution Network. Each of these NativeNet implementations has a code complexity of only 400-600 lines of C++ source code, and each took approximately one to two days to program for a single programmer. This shows the ease with which new broadcast networks can be integrated into our architecture.

The IP multicast NativeNet is implemented as described in Section 4.4. We have a similar NativeNet implementation for SSM networks. Since an SSM network can only forward single-source traffic, the BGs for this network are configured to filter all multi-source transit routes. Finally, for the CDN-based BN, the NativeNet implementation has edge servers of the CDN act as mediators to intercept requests for federation streams (in addition to local CDN streams) and forward them to the appropriate BG.

We performed a number of experiments using an experimental testbed to verify the correctness of our implementation. Our testbed consisted of four independent BNs. The BGs within the testbed were 850MHz Pentium III machines with 100Mbps network connections. We observed that the implementation maintained the right behavior in the face of changes in routing costs, breaking of peering connections, and failure of BGs.

We performed preliminary performance measurements that focus on the scalability of the architecture based on the throughput that individual BGs can sustain. We found that, due to the fact that our current implementation is a rather unoptimized prototype application running on a stock FreeBSD kernel, our experiments do not scale to very large numbers of sessions without overwhelming the machine.⁴ We are in the process of implementing a better-engineered prototype that can demonstrate substantially better scaling behavior. The point of these experiments was more to show that this approach is actually feasible.

While investigating throughput scalability, we looked at two different scenarios. In the first case, we used rate-limited sources which transmitted content at approximately 63Kbps. Figure 6 shows the average throughput (across 30 runs, each of 1 minute duration) seen per client as the number of sessions was increased from 10 to 240. We notice that

⁴Our scalability is also limited due to the fact that our single threaded implementation uses the `poll()` system call to listen for incoming data, which does not scale well with the number of sessions (i.e. number of sockets).

the throughput remains essentially constant as the number of sessions increases. This implies that the per-packet forwarding functionality within the BG is independent of the number of simultaneous sessions.

In the second case, we measure the maximum throughput that a BG can handle while varying the number of sessions. The sources are located within an HTTP-based content distribution network and they transmit data at as high a rate as can be sustained by their TCP connections. Figure 7 shows the results of that experiment. We notice that the aggregate throughput across all sessions remains constant at 82Mbps even as we increase the number of sessions. This means that each session is capable of harnessing as much of the available bandwidth as is possible given the limitations of the network and the processing power of the BGs. The aggregate throughput is limited partly because the BGs use a 100Mbps network, partly because this is a user-level implementation on an unoptimized kernel, and partly because the CPU on each BG gets saturated as it handles more and more packets per second. By using an intelligent clustered implementation for the BGs along with high-speed gigabit connections, we anticipate being able to sustain higher throughput.

6. FUTURE WORK

We are in the process of building a clustered broadcast gateway implementation in order to demonstrate better performance and scaling behavior than the single-machine implementation described above. A clustered implementation will not only allow us to incrementally scale the broadcast gateways, but also to provide better availability of each gateway. Even if one of the machines within the cluster fails, the BG will continue to operate unless the entire cluster fails.

In addition, based on our understanding of the prototype implementation, we are revising the architecture to better support collections of streams, for example, a video and audio stream that are part of the same logical session. This can be done by making explicit the notion of multiple data channels within a single session. To support this, the data-forwarding and NativeNet components will have to be aware of the existence of multiple channels and allocate communication sockets and forwarding state for each of them separately.

7. CONCLUSION

In this paper, we have argued for the need to interconnect otherwise non-interoperable broadcast-capable networks to achieve end-to-end broadcast connectivity. We have proposed an architecture that constructs a federation of broadcast networks via an overlay that is composed of application layer broadcast gateways. Our architecture can be customized to a range of network-layer and application-layer broadcast networks. Our prototype implementation demonstrates the simplicity of the customization API. The preliminary experimental results from our prototype are promising. We are currently extending our initial research in a number of directions, including building a high-performance clustered implementation of the architecture, and incorporating improved transport semantics (such as support for multi-stream sessions).

8. REFERENCES

[1] BHATTACHARYYA, S., DIOT, C., GIULIANO, L., ROCKELL, R., MEYER, J., MEYER, D., SHEPHERD, G., AND

HABERMAN, B. *An Overview of Source-Specific Multicast (SSM) Deployment*. Internet Engineering Task Force, Mar. 1999. Internet Draft, draft-ietf-ssm-overview-02.txt.

[2] BILIRIS, A., CRANOR, C., DOUGLIS, F., RABINOVICH, M., SIBAL, S., SPATSCHKE, O., AND STURM, W. *CDN Brokering*. In *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution* (Boston University, Boston, MA, June 2001).

[3] CHAWATHE, Y. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, Dec. 2000.

[4] CHAWATHE, Y., AND BREWER, E. *System Support for Scalable and Fault Tolerant Internet Services*. In *Proceedings of Middleware '98* (Lake District, U.K., Sept. 1998).

[5] CNN NEWS. Millions watch Madonna online, Nov. 2000. <http://www.cnn.com/2000/SHOWBIZ/Music/11/28/britain.madonna/>.

[6] DEERING, S., AND CHERITON, D. *Multicast Routing in Datagram Internetworks and Extended LANs*. *ACM Transactions on Computer Systems* 8, 2 (May 1990), 85–110.

[7] DEERING, S., ESTRIN, D., FARINACCI, D., JACOBSON, V., LIU, C.-G., AND WEI, L. *An Architecture for Wide-area Multicast Routing*. *IEEE/ACM Transactions on Networking* 4, 2 (Apr. 1996).

[8] DIOT, C., LEVINE, B. N., LYLES, B., KASSAN, H., AND BALENSIEFEN, D. *Deployment Issues for the IP Multicast Service and Architecture*. *IEEE Networks special issue on Multicasting* (2000).

[9] FASTFORWARD NETWORKS/INKTOMI MEDIA DIVISION. *Broadcast Overlay Architecture*, 2000. <http://www.ffnet.com/>.

[10] FENNER, B., HANDLEY, M., HOLBROOK, H., AND KOUVELAS, I. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification (Revised)*. Internet Engineering Task Force, draft-ietf-pim-sm-v2-new-05.txt, Mar. 2002. Internet Draft.

[11] FOX, A., GRIBBLE, S., CHAWATHE, Y., BREWER, E., AND GAUTHIER, P. *Cluster-based Scalable Network Services*. In *Proceedings of SOSP '97* (St. Malo, France, Oct. 1997), pp. 78–91.

[12] HUA CHU, Y., RAO, S., AND ZHANG, H. *A Case For End System Multicast*. In *Proceedings of ACM Sigmetrics '00* (Santa Clara, CA, June 2000).

[13] INTERNET ENGINEERING TASK FORCE. *IETF Content Distribution Internetworking Group*. <http://www.content-peering.org/ietf-cdi.html>.

[14] JANNOTTI, J., GIFFORD, D. K., AND JOHNSON, K. L. *Overcast: Reliable Multicasting with an Overlay Network*. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI)* (San Diego, CA, Oct. 2000), USENIX.

[15] KUMAR, S., RADOSLAVOV, P., THALER, D., ALAETTINOGLU, C., ESTRIN, D., AND HANDLEY, M. *The MASC/BGMP Architecture for Inter-domain Multicast Routing*. In *Proceedings of SIGCOMM '98* (Vancouver, British Columbia, Canada, Sept. 1998).

[16] MEYER, D., AND FENNER, B. *Multicast Source Discovery Protocol (MSDP)*. Internet Engineering Task Force, draft-ietf-msdp-spec-13.txt, Nov. 2001. Internet Draft.

[17] REKHTER, Y. *A Border Gateway Protocol 4 (BGP-4)*, Mar. 1995. RFC-1771.

[18] T. BATES, R. CHANDRA, D. KATZ, AND REKHTER, Y. *Multiprotocol Extensions for BGP-4*, Feb. 1998. RFC-2283.